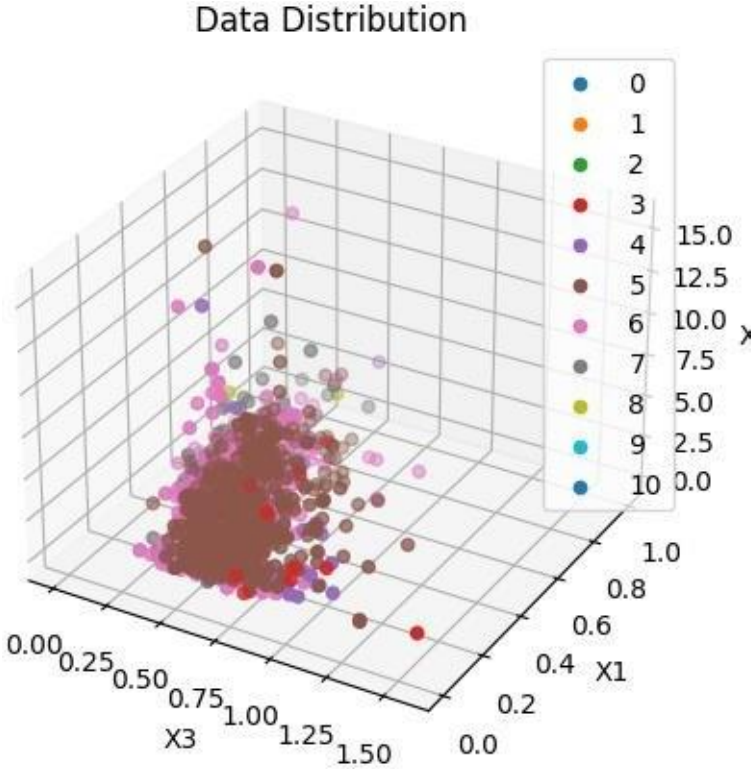
Wine Quality Classification

A. Dataset Link - https://archive.ics.uci.edu/ml/datasets/Wine+Quality. The Red Wine Dataset filewas used to train the model.



The Average Expected Risk is 0.3240594079151414

Using the sample count, the class priors are computed using the following formula:
P(L) = Numbers of samples belonging to class L / Total Number of samples

On testing the conditionality of the co-variance matrices, it is identified that majority of them yie ld a very large conditional number. Therefore, it is essential to add a small regularization value t o broaden the distribution.

$$CRegularized = CSampleAverage+\lambda I$$

In this context, λ represents a hyper-parameter that determines the level of regularization applied to the original variance values. To determine λ, I opted to compute the arithmetic mean of the ma trix's non-zero eigenvalues. The trace, or the sum of the matrix's diagonal elements, is equivalent to the sum of its eigenvalues, while the rank is the number of non-zero eigenvalues. Hence,

$$\text{Arithmetic Average} = \text{trace (CSampleAverage) / rank (CSampleAverage)}$$
$$\lambda = \alpha \text{ (Arithmetic Average)}$$

The paragraph describes a hyperparameter, represented by the symbol α, which is a real number between 0 and 1. This hyperparameter controls another hyperparameter, λ, which can be adjuste d within the range of 10-3 to 10-9 to optimize the accuracy of the model. The chosen loss functio n is the 0-1 loss, which assigns an equal penalty to all incorrect decisions and no penalty to corre ct decisions. Based on this, a classifier known as the Maximum A Posteriori (MAP) classifier ha s been designed to address the classification problem. Confusion Matrix is

Confusion Matrix is:

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         1.         0.         0.00146843
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.16981132 0.02496329
  0.01567398 0.00502513 0.         0.         0.        ]
 [0.         0.         0.         0.         0.41509434 0.56975037
  0.17711599 0.01507538 0.         0.         0.        ]
 [0.         0.         0.         0.         0.39622642 0.38913363
  0.70219436 0.53768844 0.16666667 0.         0.        ]
 [0.         0.         0.         0.         0.01886792 0.01468429
  0.0846395  0.36180905 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.02037618 0.08040201 0.83333333 0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]]
```

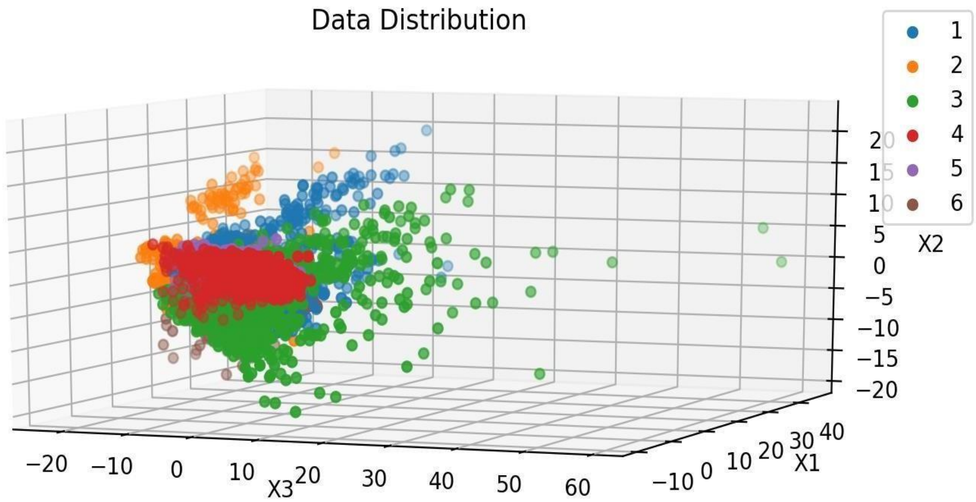
Human Activity Classification

Dataset Links –

https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones
https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones

The given information states that there are 561 features, which is a substantial number, but itcomes with some drawbacks, such as some features being redundant and not contributing to classification, and correlated features providing similar information. Additionally, it takes more computation time. Consequently, to mitigate these issues, PCA is employed as a technique for reducing dimensionality. It transforms the features into a smaller number of relevant ones by maximizing the variance.

Data Visualization after applying PCA-



From the data distribution and class priors calculated, it is evident that the data is evenly distributed between all the classes. The regularization parameter and assumptions regarding class-conditional PDF are similar to the previous part.

$$
\mathbf{C} = \begin{matrix}
\mathbf{0.912} & \mathbf{0.023} & \mathbf{0.029} & \mathbf{0} & \mathbf{0.0014} & \mathbf{0} \\
\mathbf{0.056} & \mathbf{0.919} & \mathbf{00.125} & \mathbf{0.0015} & \mathbf{0} & \mathbf{0} \\
\mathbf{0.021} & \mathbf{0.0567} & \mathbf{0.845} & \mathbf{0} & \mathbf{0} & \mathbf{0.0007} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0.604} & \mathbf{0.08} & \mathbf{0.032} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0.375} & \mathbf{0.915} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0.0398} & \mathbf{0.002} & \mathbf{0.967}
\end{matrix}
$$

Minimum Expected risk is 0.091.

The Wine Quality Classification problem is better solved using a Gaussian classifier, whichproduces more accurate results. This is because the dataset is balanced, meaning that the class priors are similar. As a result, the class posteriors depend largely on the class- conditional distributions, and the model can establish correlations between the reduced features and class labels. While principal component analysis (PCA) is a useful method for extracting significant information from the extensive feature set, it is important to evaluate the model on new samples to avoid overfitting to the training set.

Please find the below GitHub Link for the code:

GitHub Link

Appendix:

A

```
import
random
import
numpy as
np import
numpy.m
atlib
import matplotlib.pyplot as plt
from scipy.stats import
multivariate_normalfrom
mpl_toolkits.mplot3d import
Axes3D import pandas as pd
from numpy import linalg as LA

# compute mean and covariance
def compute_mean_and_covariance(No_samples, No_features, class_labels,
    No_labels):unq_ls = len(np.unique(class_labels))
    mean = np.zeros( [No_labels, No_features] )
    covariance = np.zeros( [No_labels, No_features, No_features] )

#     for i, cls_i in enumerate(np.unique(class_labels)):
#        mean[i, :] = np.mean(df[df['quality'] == cls_i].drop(columns=['quality']), axis = 0)
#      covariance[i, :, :] = np.cov(df[(df['quality'] == cls_i)].drop(columns=['quality']), rowvar =
False)
#        covariance[i, :, :] += (0.00000005) *
((np.trace(covariance[i, :, :])) /LA.matrix_rank(covariance[i, :,
:])) * np.eye(No_features)
    for i in range(No_labels):
        mean[i, :] = np.mean(df[df['quality'] == i].drop(columns=['quality']), axis = 0)

        if (i not in class_labels):
            covariance[i, :, :] =
        np.eye(No_features)else:
            covariance[i, :, :] = np.cov(df[df['quality'] == i].drop(columns=['quality']),
            rowvar = False)covariance[i, :, :] += (0.00000005) * ((np.trace(covariance[i, :,
:]))/LA.matrix_rank(covariance[i, :, :])) *

    np.eye(No_features)return mean, covariance

# compute class condtional pdf
def compute_class_conditional_pdf(class_labels, No_labels,
No_samples, mean,covariance_matrix):
    P_x_given_L = np.zeros(shape = [No_labels,
    No_samples])unq_ls = np.unique(class_labels)
    for i in unq_ls:
        P_x_given_L[i, :] = multivariate_normal.pdf(df.drop(columns =
['quality']), mean[i, :],covariance_matrix[i, :,:])
```

```
    return P_x_given_L

# compute class priors based on sample count
def class_priors(No_labels, class_labels,
    No_samples):priors = np.zeros(shape =
    [11, 1])
    for i in range(0, No_labels):
        priors[i] = (np.size(class_labels[np.where((class_labels == i))])) /
    No_samplesreturn priors

# compute Confusion Matrix
def compute_confusion_matrix (No_labels,
    class_labels):cm = np.zeros(shape =
    [No_labels, No_labels])
    for i in
        range(No_lab
        els): for j in
        range(No_lab
        els):
            if j in class_labels and i in class_labels:
                cm[i, j] = (np.size(np.where((i == Decision) & (j ==
class_labels)))) /np.size(np.where(class_labels == j))
    return cm

# Import Dataset
df = pd.read_csv(r'C:\Users\Shiva Kumar
Dande\Downloads\winequality-red.csv')df = df.dropna()
Data =
df.to_num
py()
#print(Dat
a)
No_Classes = 11

rows, columns =
np.shape(df)
No_samples =
rows No_features
= columns - 1
No_labels = 11
df.dropna(inplace
=True)
class_labels = df['quality']
class_labels =
np.array(class_labels)

# Plot Data
Distribution
fig =
plt.figure()
```

```
ax =
plt.axes(projection
= "3d")for i in
range(No_labels):
    ax.scatter(Data[(class_labels==i),1],Data[(class_labels==i),4],Data[(class_labels==i),8], label =
i)
plt.
xla
bel(
'X3
')
plt.yla
bel('X1
')
ax.set_
zlabel('
X2')
ax.lege
nd()
plt.title('Data
Distribution')
plt.show()

loss_matrix = np.ones([No_labels, No_labels]) - np.eye(No_labels)
mean, covariance = compute_mean_and_covariance(No_samples, No_features,
class_labels,No_labels)
pdf = compute_class_conditional_pdf(class_labels, No_labels, No_samples, mean,
covariance)p = class_priors(No_labels, class_labels, No_samples)

# Compute Class Posteriors using priors and class
conditional PDFP_x = np.matmul(np.transpose(p), pdf)
class_posteriors = (pdf * (np.matlib.repmat(p, 1, No_samples))) / np.matlib.repmat(P_x,
No_labels,1)

# Evaluate Expected risk and decisions based on
minimum riskexpected_risk =
np.matmul(loss_matrix, class_posteriors) Decision
= np.argmin(expected_risk, axis = 0)
avg_exp_risk = np.sum(np.min(expected_risk, axis = 0)) /
No_samplesprint(f'The Average Expected Risk is
{avg_exp_risk}')
cm = compute_confusion_matrix (No_labels,
class_labels)print(cm)
```

B

```
import
pandas as
pd import
random
import
numpy as
np import
numpy.m
atlib
import matplotlib.pyplot as plt
from scipy.stats import
multivariate_normalfrom
mpl_toolkits.mplot3d import
Axes3D from numpy import
linalg as LA
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

def compute_mean_and_covariance(No_samples, No_features, class_labels,
    No_labels):mean = np.zeros(shape = [No_labels, No_features])
    covariance = np.zeros(shape = [No_labels, No_features, No_features])
    for i in range(0, No_labels):
        mean[i, :] = np.mean(Data[(class_labels == i + 1), :], axis = 0)
        covariance[i, :, :] = np.cov(Data[(class_labels == i + 1), :],
        rowvar = False)
        covariance[i, :, :] += (0.00001) *
        ((np.trace(covariance[i,:,:]))/LA.matrix_rank(covariance[i,:,:]))
* np.eye(10)
        #Check if covariance matrices are ill-
        conditioned
        #print(LA.cond(covariance_matrix[i,:
        ,:]))
    return mean, covariance

def compute_class_conditional_pdf(class_labels, No_labels, No_samples, mean,
    covariance_matrix):P_x_L = np.zeros(shape = [No_labels, No_samples])
    for i in range(0, No_labels):
        P_x_L[i, :] = multivariate_normal.pdf(Data, mean[i, :],
    covariance[i, :,:])return P_x_L

def class_priors(No_labels, class_labels,
    No_samples):p = np.zeros(shape =
    [No_labels, 1])
    for i in range(0, No_labels):
        p[i] = (np.size(label[np.where((class_labels == i + 1))])) /
    No_samplesreturn p

def compute_confusion_matrix (No_labels,
    class_labels):cm = np.zeros(shape =
```

```
    [No_labels, No_labels])
    for d in
        range(No_labe
        ls): for l in
        range(No_labe
        ls):
            cm[d, l] = (np.size(np.where((d == Decision) & (l ==
class_labels - 1)))) /np.size(np.where(class_labels - 1 == l))
    return cm


df = pd.read_csv(r'C:\Users\Shiva Kumar Dande\Desktop\Machine Learning
Projects\Assignment1\HumanDetection\X_train.txt')
Data = df.to_numpy()

No_samples = Data.shape[0]
Y = pd.read_csv(r'C:\Users\Shiva Kumar Dande\Desktop\Machine Learning
Projects\Assignment1\HumanDetection\y_train.txt')
class_labels = np.squeeze(Y.to_numpy())

Data =
Data[:, 0:-
2] sc =
StandardS
caler()
Data = sc.fit_transform(Data)

pca =
PCA(n_components
= 10)Data =
pca.fit_transform(Da
ta)

No_labels = 6           # Number
of labels No_features = 10
                        # Number
of features

# Plot Data
Distribution
fig =
plt.figure()
ax =
plt.axes(projection
= "3d")for i in
range(1, N_labels +
1):
    ax.scatter(Data[(class_labels==i),1],Data[(class_labels==i),2],Data[(class_lab
els==i),3], class_labels=i)
plt.xlabel('X3')
```

```
plt.ylabel('X1') ax.set_zlabel('X2')ax.legend()
plt.title('Data
Distribution')
plt.show()

loss_matrix = np.ones(shape = [No_labels, No_labels]) - np.eye(No_labels)
mean, covariance = compute_mean_and_covariance(No_samples, No_features,
class_labels,No_labels)
pdf = compute_class_conditional_pdf(class_labels, No_labels, No_samples, mean,
covariance)p = class_priors(No_labels, class_labels, No_samples)

P_x = np.matmul(np.transpose(priors), P_x_given_L)
ClassPosteriors = (P_x_given_L * (np.matlib.repmat(priors, 1, N))) /
np.matlib.repmat(P_x, N_labels,1)
#expected risk
er = np.matmul(loss_matrix,
ClassPosteriors)Decision =
np.argmin(er, axis = 0)

print("Average Expected Risk", np.sum(np.min(er, axis = 0)) /
No_samples)cm = compute_confusion_matrix (No_labels,
class_labels) np.set_printoptions(suppress=True)
print(cm)
```